Grid Troubleshooting and Performance Analysis

Brian L. Tierney

BLTierney@lbl.gov

Distributed Systems Department
Lawrence Berkeley National Laboratory

The Problem

- Assume a Grid job is:
  - submitted to a resource broker, uses a reliable file transfer service to copy several files, then runs the job.
- This normally takes 15 minutes to complete. But…
  - two hours have passed and the job has not yet completed.
- What, if anything, is wrong?
  - Is the job still running or did one of the software components crash?
  - Is the network particularly congested?
  - Is the CPU particularly loaded?
  - Is there a disk problem?
  - Was a software library containing a bug installed somewhere?

The Solution

- End-to-End Monitoring
  - All components between the application endpoints must be monitored.
- This includes:
  - software (e.g., applications, services, middleware, operating systems)
  - end-host hardware (e.g., CPUs, disks, memory, network interface)
  - networks (e.g., routers, switches, or end-to-end paths)

Monitoring Components

- A complete End-to-End monitoring framework includes:
  - Instrumentation Tools
    - Facilities for precision monitoring of all software (applications, middleware, and operating systems) and hardware (host and network) resources
  - Monitoring Data Publication
    - Standard schemas, discovery and publication mechanisms, and access policies for monitoring event data are required
  - Sensor Management
    - The amount of monitoring data produced can quickly become overwhelming
    - A mechanism for activating sensors on demand is required
  - Data Analysis Tools
    - event analysis and visualization tools
  - Event Archives
    - Historical data used to establish a baseline to compare current and predict future performance

Uses for Monitoring Data

- Troubleshooting and Fault Detection
  - Detect failures and recovery
- Performance analysis and tuning
  - Better program design (e.g.: will better pipelining of I/O and computation help?)
  - Network-aware Applications (TCP buffer size tuning, # of parallel streams, etc.)
- Debugging
  - Complex, multithreaded, distributed programs are difficult to debug without the proper monitoring data
- Guiding scheduling decisions
  - Grid Schedulers
    - Find the best match of CPUs and data sets for a given job
  - Grid Replica Selection
    - Find the “best” copy of a data set to use
- Auditing and intrusion detection

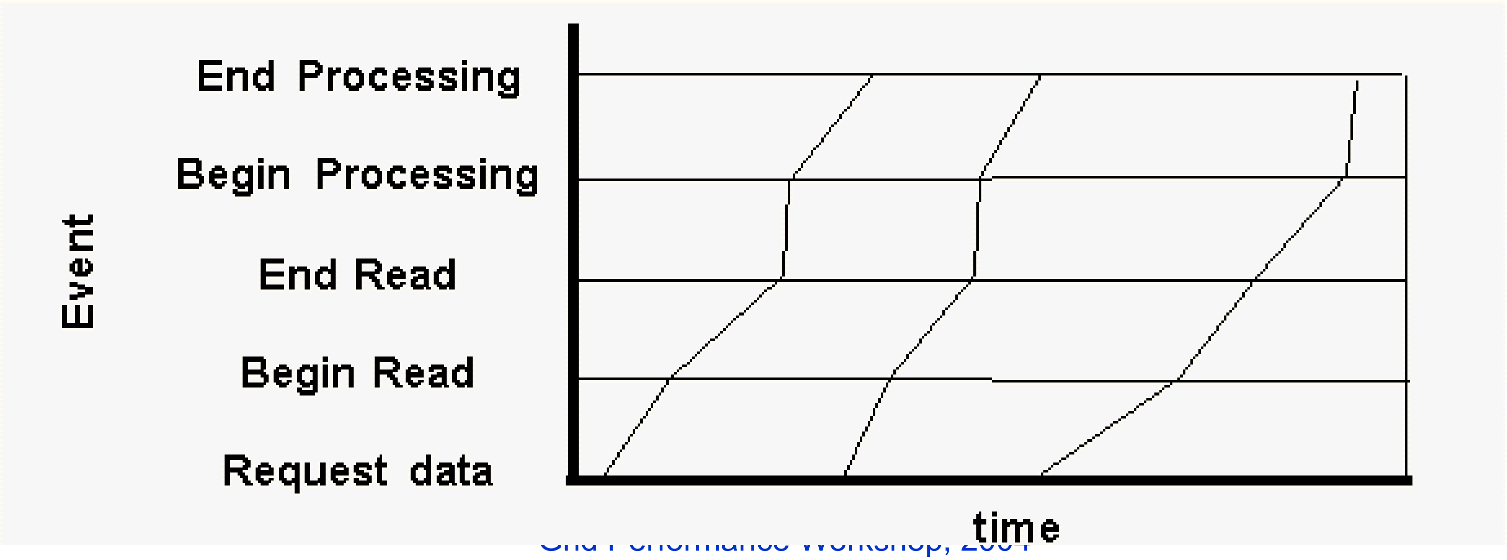
Instrumentation: NetLogger Toolkit

NetLogger Toolkit

- We have developed the NetLogger Toolkit (short for **Net**worked Application **Logger**), which includes:
  - tools to make it easy for distributed applications to log interesting events at every critical point
    - NetLogger client library (C, C++, Java, Perl, Python)
    - Extremely light-weight: can generate > 900,000 events / second on current systems (9000 events / sec with 1% app. perturbation)
  - tools for host and network monitoring
  - event visualization tools that allow one to correlate application events with host/network events
  - NetLogger event archive and retrieval tools
- NetLogger combines network, host, and application-level monitoring to provide a complete view of the entire system.

NetLogger Analysis: Key Concepts

- NetLogger visualization tools are based on time correlated and object correlated events.
  - precision timestamps (default = microsecond)
- If applications specify an "object ID" for related events, this allows the NetLogger visualization tools to generate an object "**lifeline**"
- In order to associate a group of events into a "lifeline", you must assign an "object ID" to each NetLogger event
  - Sample Event ID: file name, block ID, frame ID, Grid Job ID, etc.



Sample NetLogger Instrumentation

```
log = netlogger.open(“x-netlog://log.lbl.gov”,”w”)
done = 0
while not done:
   log.write(0,"EVENT_START","TEST.SIZE=%d",size)
   # perform the task to be monitored
   done = do_something(data,size)
   log.write(0,"EVENT_END")
```

- Sample Event:

```
DATE=20000330112320.957943 HOST=gridhost.lbl.gov \
PROG=gridApp LVL=1 NL.EVNT=WriteData SEND.SZ=49332
```

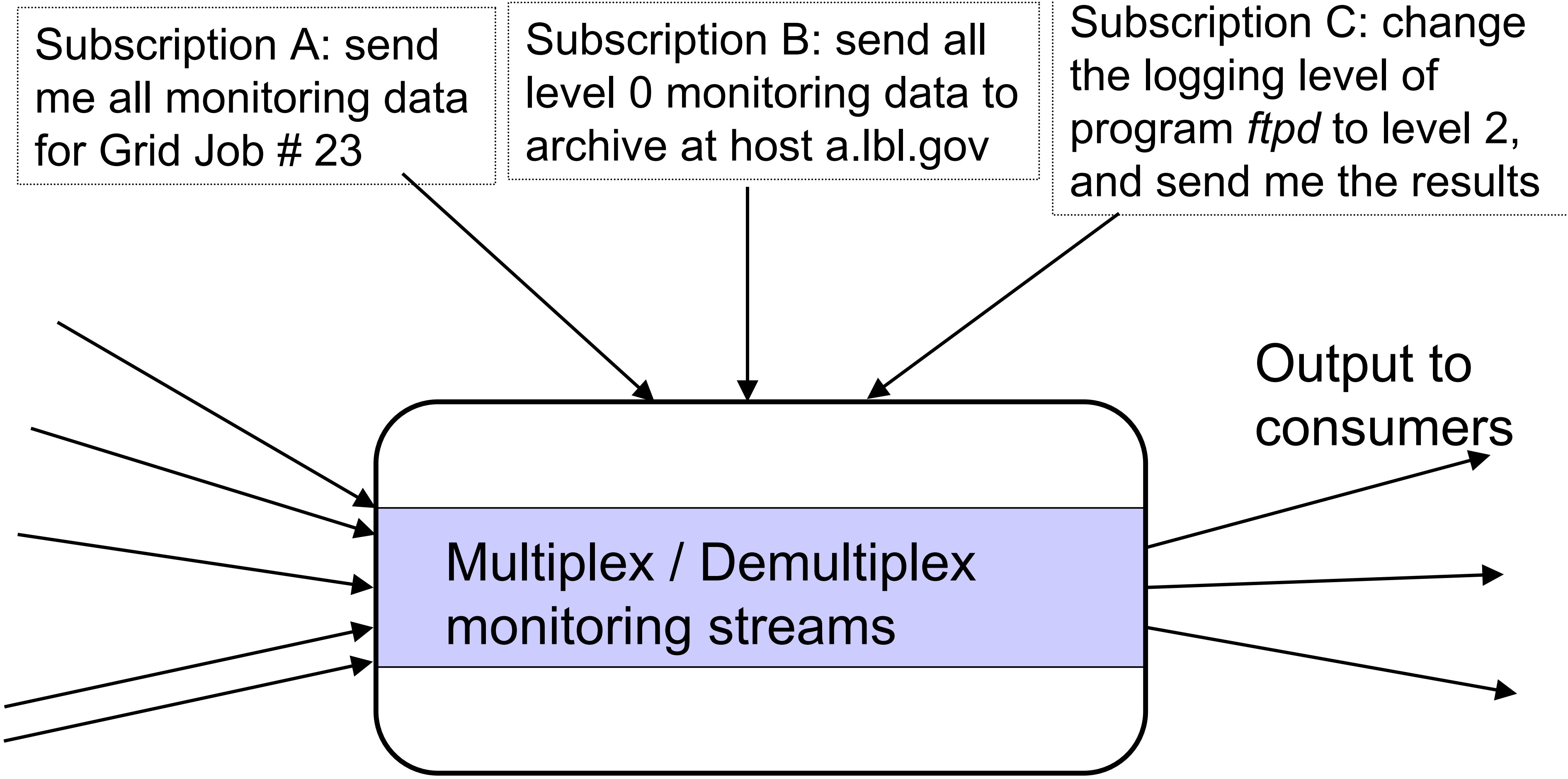
NetLogger Tools

- nlforward: Log file forwarder
  - forwards a single NetLogger file or directory of files to an output URL
- netlogd: TCP socket server daemon
  - accepts one or more NetLogger TCP streams and writes them to one or more NetLogger output URL's

NetLogger Activation Service

- Do not want all monitoring data collected all the time
  - Potentially way too much data
  - Need to adjust the level of monitoring as needed for:
    - Debugging
    - Performance tuning
    - Error analysis
- NetLogger Activation Service addresses this issue
  - NetLogger-based sensors register with the activation service
  - Very useful debugging tool for MPI / PC cluster-based jobs

NetLogger Filter and Activation Service



Subscription A: send me all monitoring data for Grid Job # 23

Subscription B: send all level 0 monitoring data to archive at host a.lbl.gov

Subscription C: change the logging level of program *ftpd* to level 2, and send me the results

Output to consumers

Multiplex / Demultiplex monitoring streams

Incoming monitoring data:

- application,
- middleware,
- host

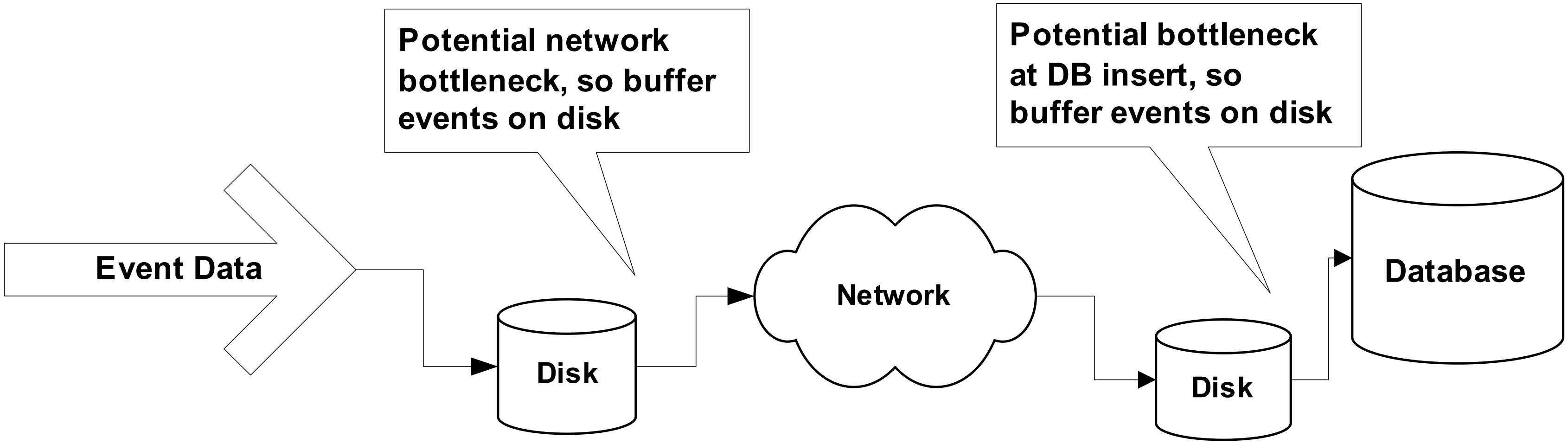
NetLogger Filter and Activation Service

Relational Event Archive

- A relational database that supports SQL is an excellent tool for data mining of event data.
  - SQL provides a general and powerful language for extracting data.
- For example, with SQL we can easily do queries such as:
  - find the average throughput for the past 100 runs
  - return all events for application runs that coincided with reports of network errors
  - return all events for application runs where the throughput dropped below 10 Mbits/sec and CPU load was over 90%
  - return all events for applications runs where the total execution time was more than 50% above the average time for the past month

NetLogger Archive Architecture

- Architecture must be scalable and capable of handling large amounts of application event data,
- None of the components can cause the pipeline to “block” while processing the data, as this could cause the application to block
    - For example, instrumented FTP server could send > 6000 events/second to the archive (500 KB/sec (1.8 GB/hr) of monitoring event data)


Potential network bottleneck, so buffer events on disk
Potential bottleneck at DB insert, so buffer events on disk
Event Data
Disk
Network
Disk
Database

Grid Troubleshooting Example

- Step 1: insert instrumentation code during the development stage
  - to ensure the program is operating as expected
- Step 2: establish a performance baseline for this service, and store this information in the monitoring event archive.
  - Include system information such as processor type and speed, OS version, CPU load, disk load, network load data, etc.
- Step 3: put service into production, and everything works fine
  - Until…..
- One day, users start complaining that service X is taking much longer than previously

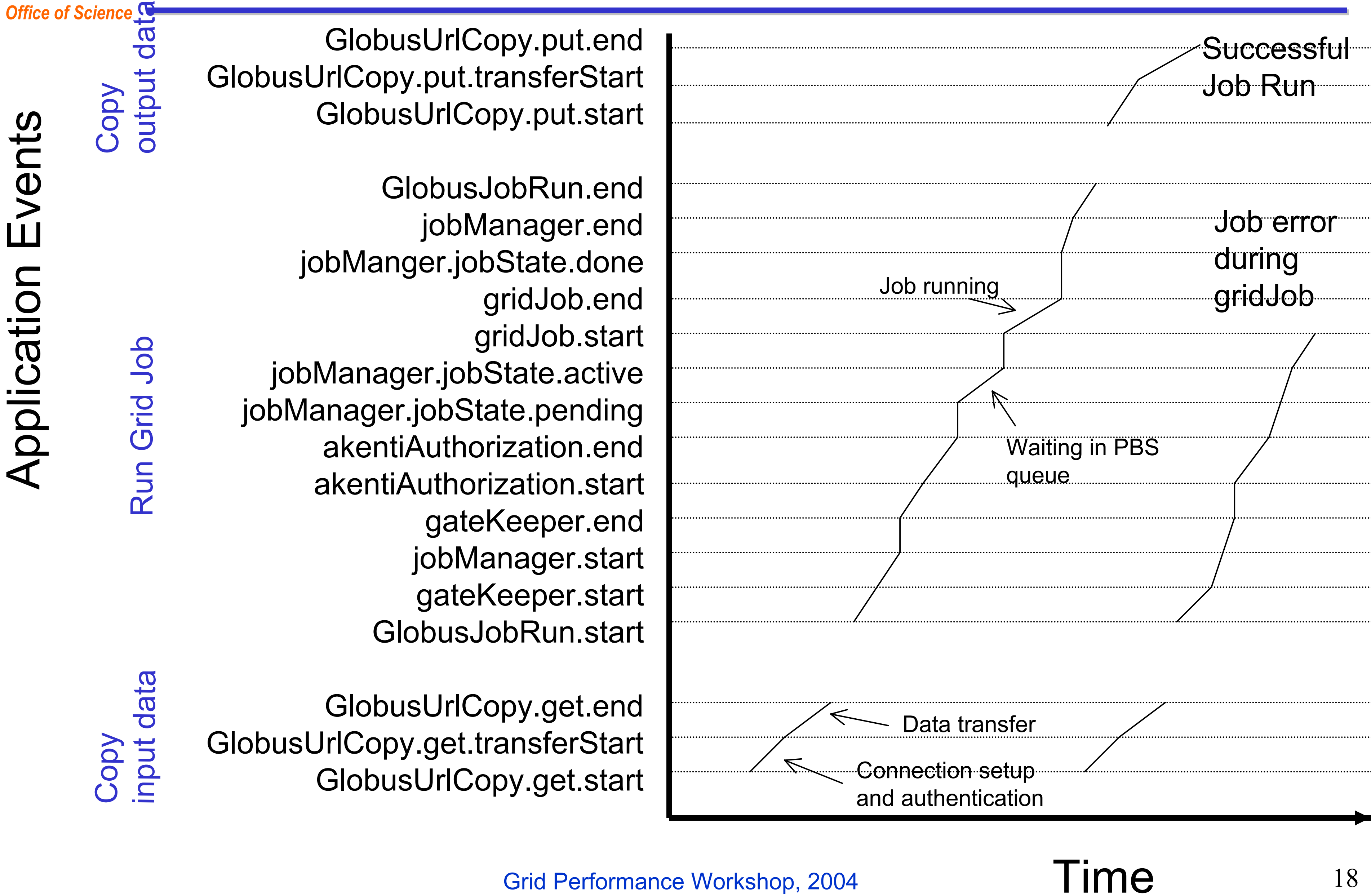
Grid Troubleshooting Example

- To collect data for analysis, one must:
  - Locate relevant monitoring data, and subscribe to that data.
  - Activate any missing sensors, and subscribe to the their data.
  - Activate debug-level instrumentation in the service, and subscribe.
  - Locate monitoring data in the monitoring event archive for the baseline test from when things were last working.
- Data analysis can be then begin:
  - Check the hardware and OS information to see if anything changed.
  - Look at the application instrumentation data to see if anything looks unusual.
  - Look at the system monitoring data to see of anything looks unusual (e.g., unusually high CPU load).
  - Correlate the application and middleware instrumentation data with the host and network monitoring data.

Grid Job ID

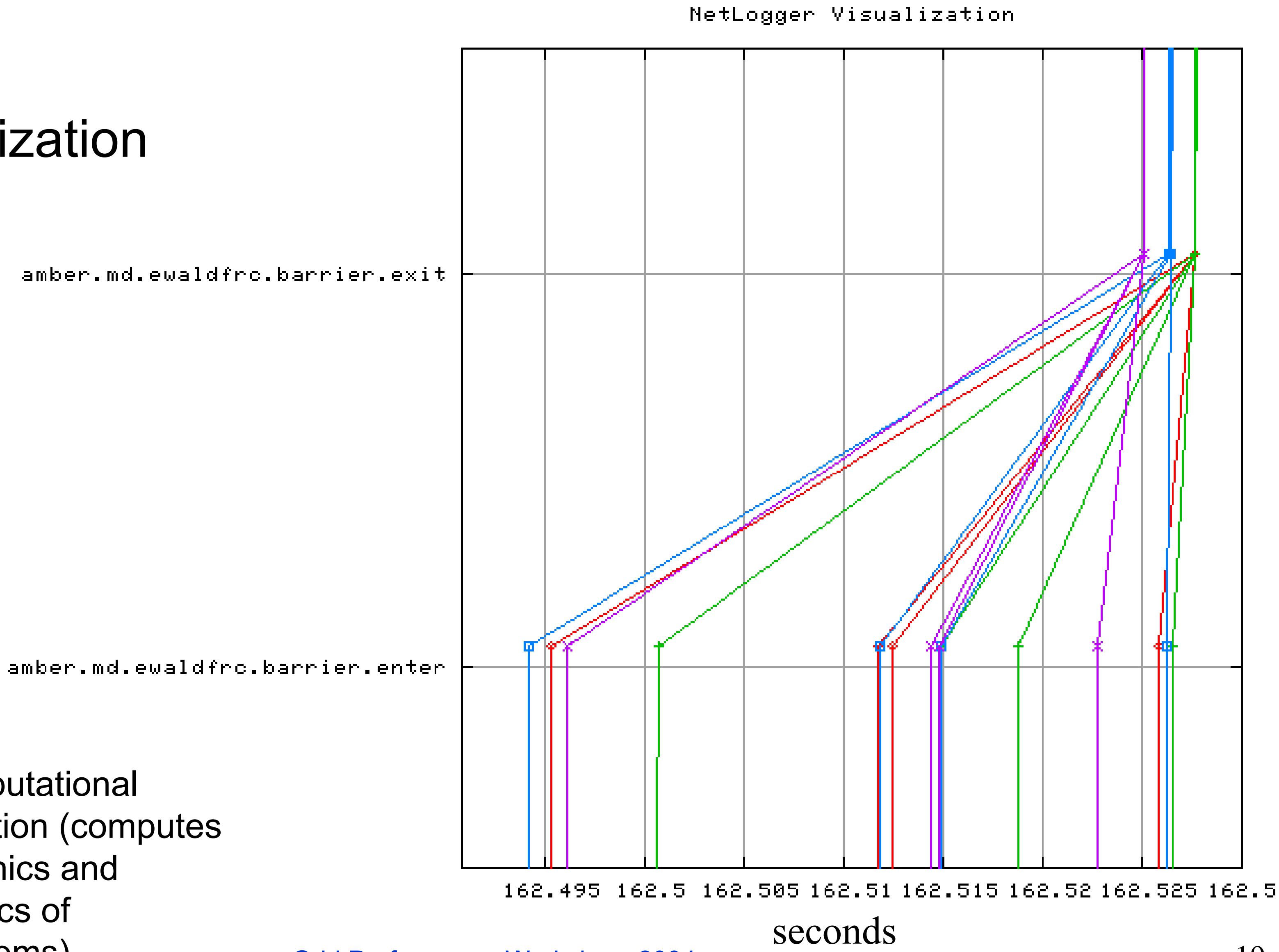
- In order to graphically link events from several Grid components
  - monitoring events for the same “job” needs the same “Grid Job ID” (GID)
- We have instrumented the following pyGlobus components with NetLogger with a GID
  - globus-job-run, globus-url-copy, Globus gatekeeper Globus job manager
  - globus-job-run generates the GID using *uuidgen*
  - GID passed to gatekeeper via RSL
  - In WSRF-based Grids, it should be possible to standardize a mechanisms to pass GID’s between Grid Services

Troubleshooting Example:
Step 1: Generate Grid Job “Lifeline”



Step 2: Add detailed application instrumentation

e.g.: MPI Synchronization Barrier



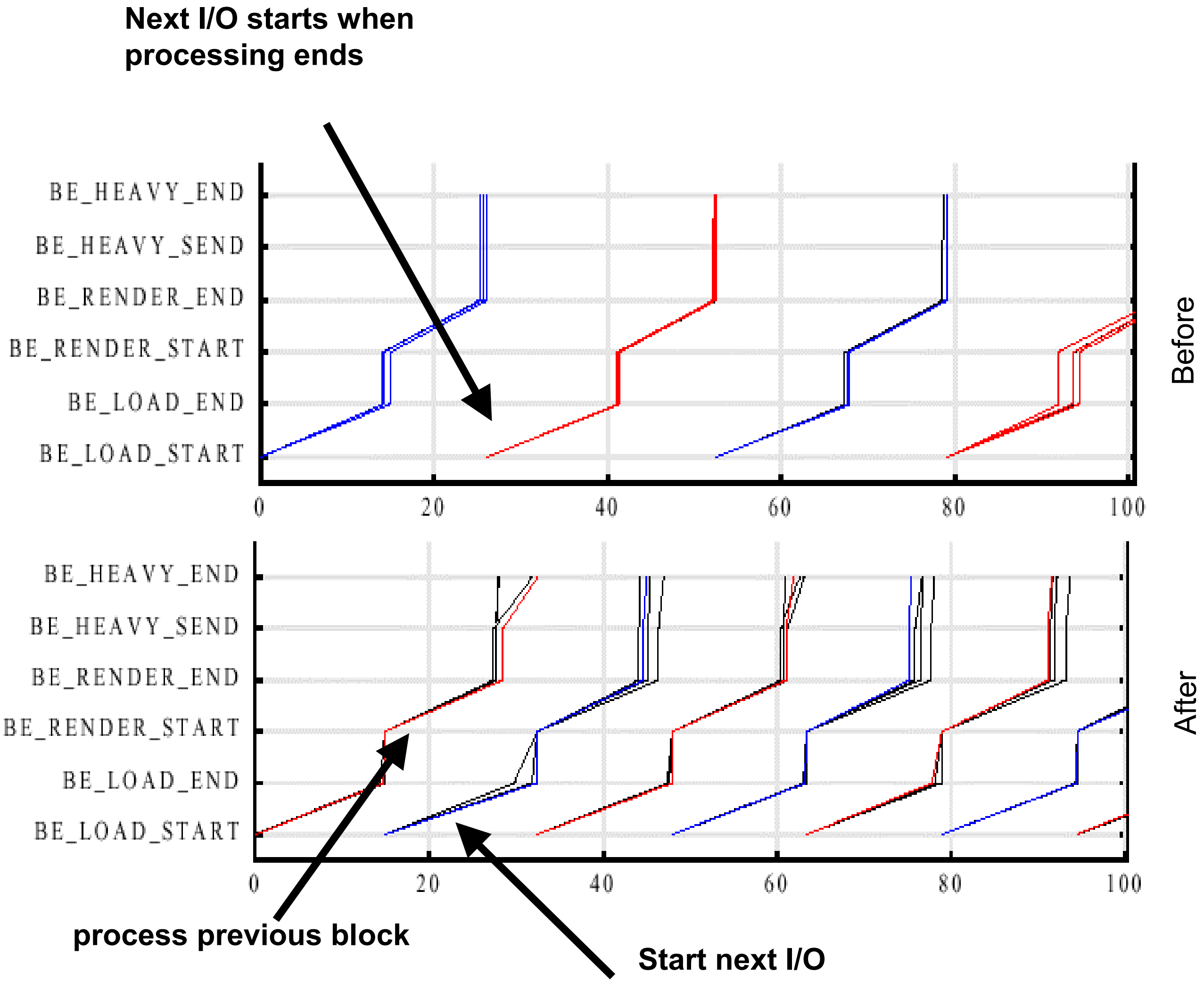
AMBER is a computational chemistry application (computes molecular mechanics and molecular dynamics of biomolecular systems)

Step 2: Add detailed application instrumentation, 2nd example

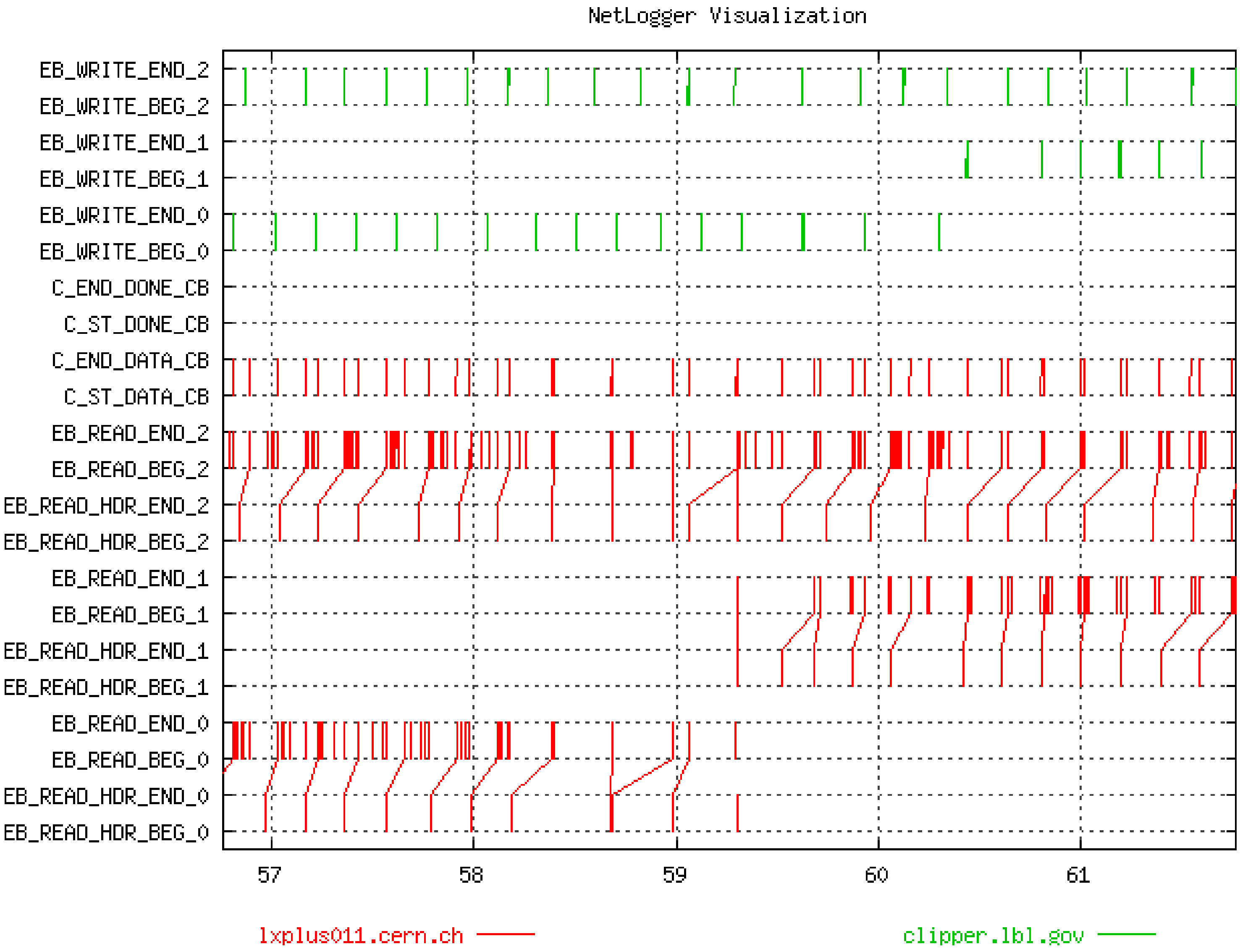
- I/O followed by processing

- overlapped I/O processing

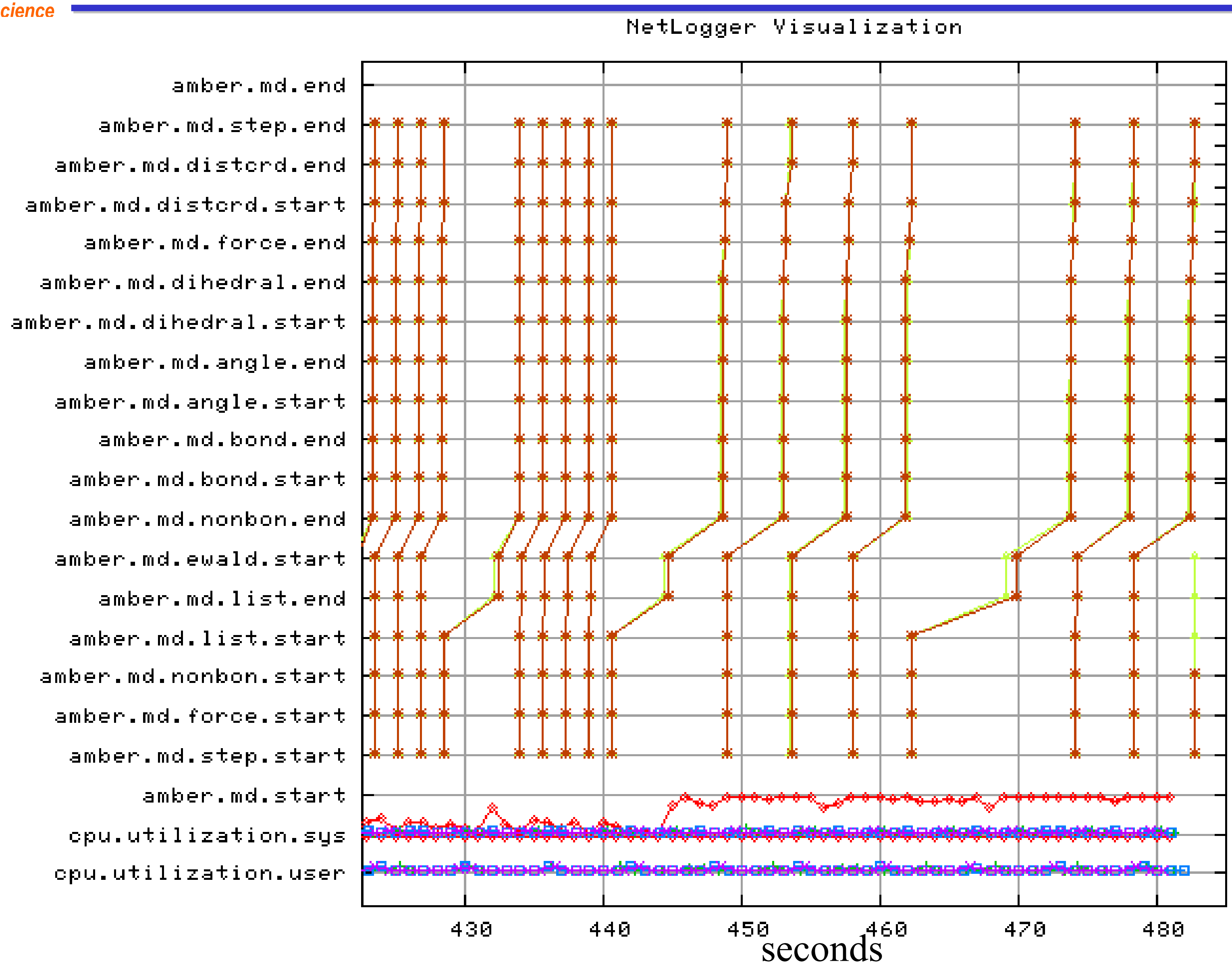
almost a 2:1 speedup



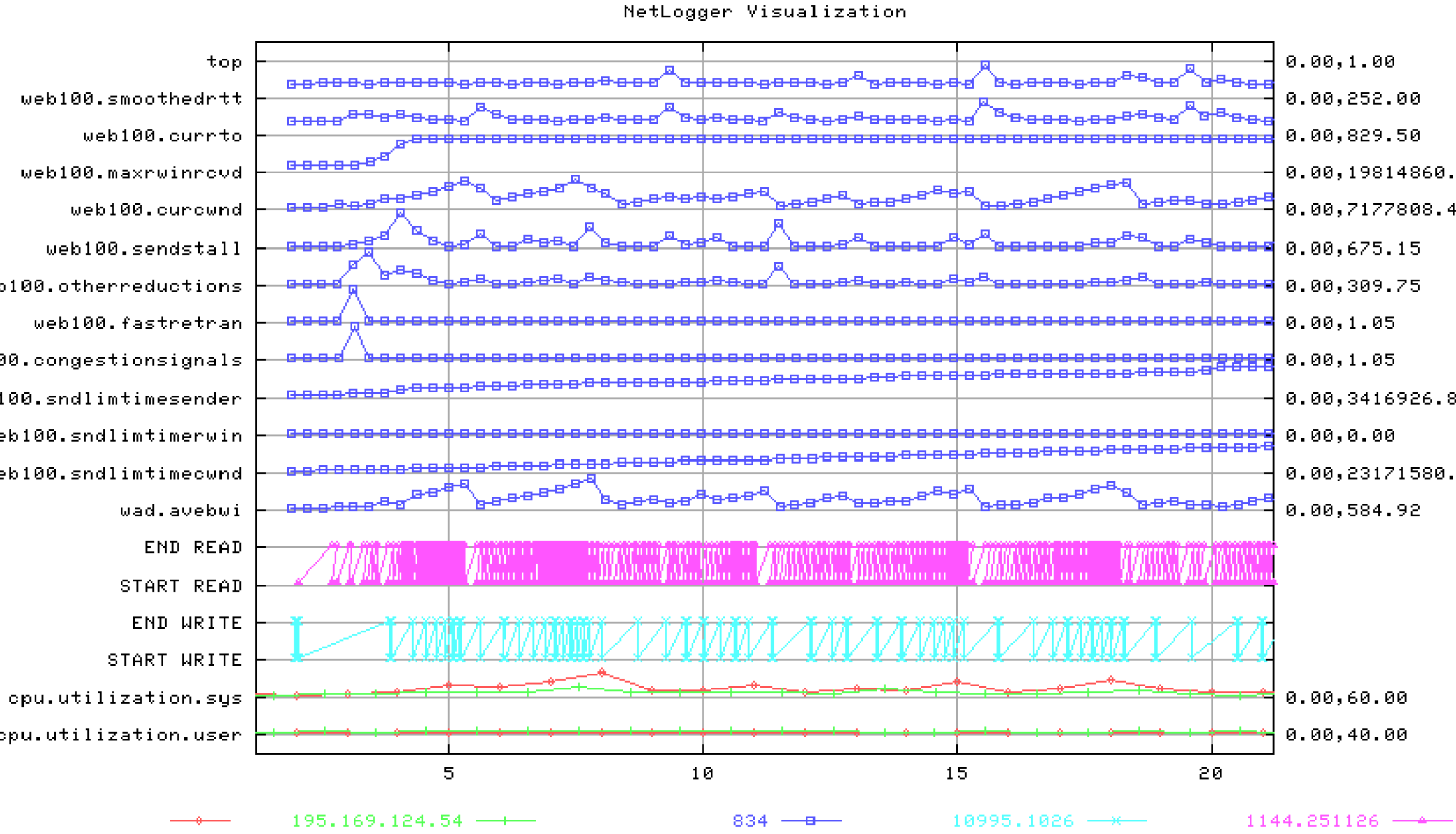
Step 2: 3rd example
Parallel Stream Bug



Step 3: add host monitoring
(e.g.: CPU load or TCP retransmits)



Step 4: add TCP monitoring

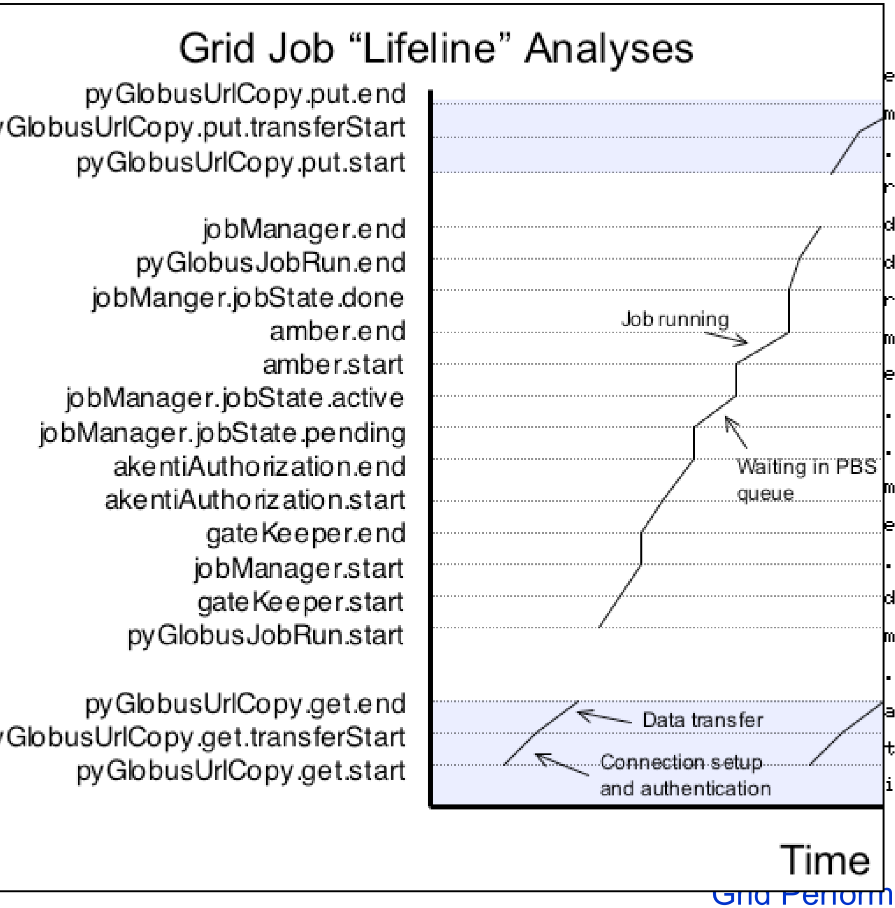


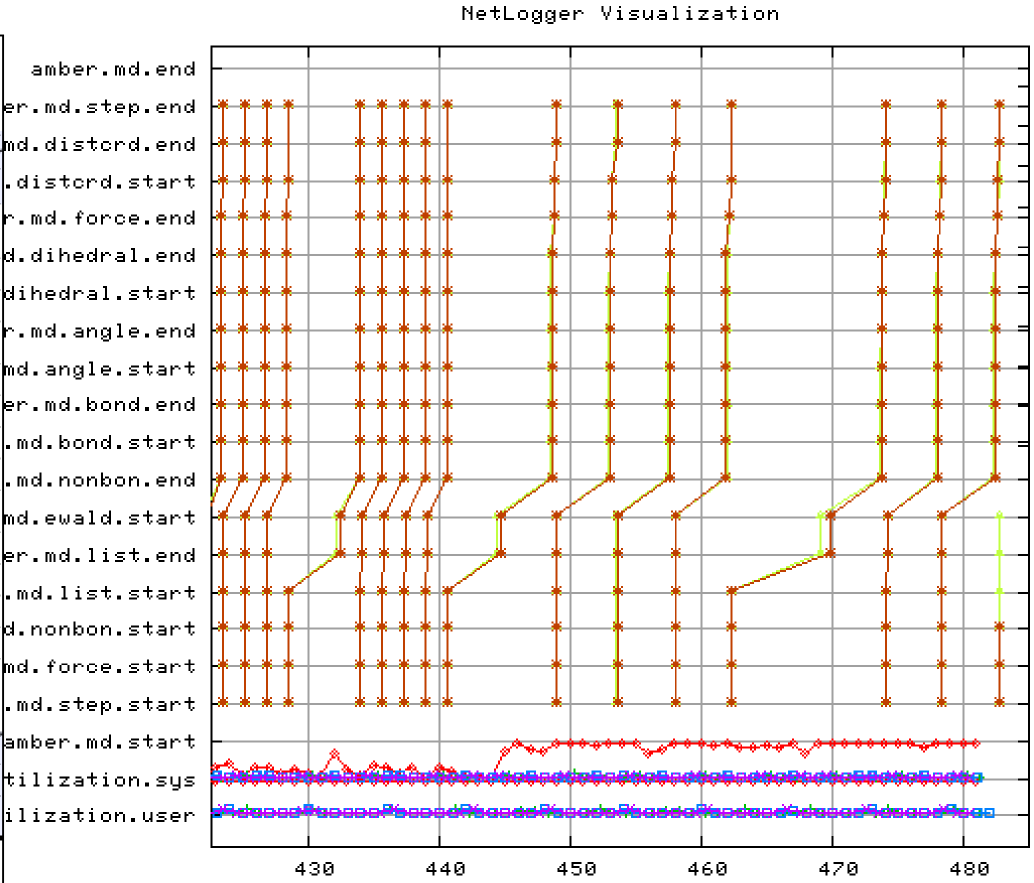
Conclusions

- End-to-end, top-to-bottom instrumentation is essential
  - Need ability to “drill down” from high level view into each component
- Instrumenting and benchmarking user applications is essential
  - Often applications are the source of the problems

Conclusions

- Grid ID is essential for correlating events





Open Issues

- How to determine general-purpose performance baselines?
- Anomaly detection
  - Borrow some ideas from the Intrusion detection community?
- How to propagate "Grid IDs" to all components?

For More Information

- DMF: http://dsd.lbl.gov/NetLogger/
  - All software components are available for download under DOE/LBNL open source license (BSD-style)

- email: BLTierney@LBL.GOV